Database

Data is not stored separately but rather in fixed locations. Convert tables into database-related formats. A database is an organized collection of data that allows users to store, retrieve, and manage data. Database systems make data storage and management more efficient, reliable, and convenient.

Basic Concepts of Databases: **Database Management System (DBMS)**

A database management system is software for managing databases. We will create data, read data, update data, and delete data, also known as CRUD operations, to ensure data consistency, integrity, and security.

Relational Database:

A relational database is a common type of database that is essentially a table. Each table contains rows and columns. Columns represent data attributes. In a user table, fields can include username, password, email, etc. Rows represent individual data entities. In the user table, a row can represent a single user's information.

Three key concepts:

- **Primary Key:** It is a core concept and is a unique identifier for each record in the table. For instance, the user ID in a user table can serve as the primary key.

Suppose there is a user table `Users`:

User ID	Username	Password	Email
1	Alice	password1	alice@example.com
2	Bob	password2	bob@example.com
3	Charlie	password3	charlie@example.com

Here, `User ID` is the primary key because it uniquely identifies each user.

- **Foreign key:** used to establish a relationship with another table. The foreign key value must match the primary key value in the other table.

Suppose there is an order table `Orders`:

Order ID	User ID	Product	Quantity
101	1	Laptop	1
102	2	Smartphone	2
103	1	Tablet	1

Here, `User ID` is a foreign key, referencing the `User ID` in the user table `Users`. This indicates that each order is created by a user in the user table.

- The last concept is query. **Query** is the operation of checking data from the database. For example, if I select the user with ID 208, I am querying this data.

Example:

Suppose you want to query the user information with User ID 1 from the user table, you can use the SQL query: **SELECT * FROM Users WHERE User ID = 1;**

```
| User ID | Username | Password  | Email             |
|---------|----------|-----------|-------------------|
| 1       | Alice    | password1 | alice@example.com |
```

Common database management systems:

- MySQL is an open-source relational database management system
- PostgreSQL is also a database relational system
- Oracle Database
- Microsoft SQL

```
import pandas as pd

file_path = 'C:/Users/Frank/Desktop/Updated_DHT11_Data_Collection_v1.xlsx' #
Define the path of the Excel file
xls = pd.ExcelFile(file_path) # Load the Excel file into Pandas' ExcelFile object

sheet_names = xls.sheet_names # Get the sheet names in the Excel file
```

```
df = pd.read_excel(file_path, sheet_name='Sheet1') # Load the first sheet of the
Excel file into a DataFrame

df['Timestamp'] = pd.to_datetime(df['Timestamp']) # Convert the 'Timestamp'
column to datetime format

# Create the sensors table

sensors = df[['Sensor ID','Location']].drop_duplicates().reset_index(drop=True)

sensors['Sensor ID'] = sensors['Sensor ID'].astype(str)
sensors['Location'] = sensors['Location'].astype(str)
sensors['Sensor Key'] = sensors.index + 1

# Create the Measurements table

measurements = df.drop(columns=['Location']).copy()
measurements = measurements.merge(sensors, on='Sensor ID', how='left')
measurements = measurements.drop(columns=['Sensor ID'])
measurements = measurements.rename(columns={'Sensor Key':'Sensor ID'})

# Group by 'TimeZone' and calculate the mean and standard deviation of
'Temperature' and 'Humidity'

grouped = measurements.groupby('TimeZone').agg({
    'Temperature': ['mean', 'std'],
    'Humidity': ['mean', 'std']
}).reset_index()

# Flatten the MultiIndex columns and rename them for easier understanding
grouped.columns = ['TimeZone', 'Mean Temperature', 'Temperature Std Dev',
                   'Mean Humidity', 'Humidity Std Dev']

# Calculate the change rate of 'Temperature' and 'Humidity' and convert to
percentage
measurements['Temperature Change Rate'] =
measurements['Temperature'].pct_change() * 100
measurements['Humidity Change Rate'] = measurements['Humidity'].pct_change() *
100

# Calculate the comfort index with the formula: Temperature - Humidity * 0.55 +
(Temperature - mean Temperature) * 0.1
```

```
measurements['Comfort Index'] = measurements['Temperature'] -
measurements['Humidity'] * 0.55 + (measurements['Temperature'] -
measurements['Temperature'].mean())*0.1

# Replace 0 values in temperature change rate with 'no change'
measurements['Temperature Change Rate'] = measurements['Temperature Change
Rate'].apply(
    lambda x: 'no change' if x == 0 else x
)

# Replace 0 values in humidity change rate with 'no change'
measurements['Humidity Change Rate'] = measurements['Humidity Change
Rate'].apply(
    lambda x: 'no change' if x == 0 else x
)

measurements['Comfort Level'] = pd.cut(measurements['Comfort Index'], bins=[-
float('inf'),
                                                                        measu
rements['Comfort Index'].quantile(0.33),
                                                                        measu
rements['Comfort Index'].quantile(0.66), float('inf')],
                                                                        label
s=['Low', 'Medium', 'High'])

output_file_path_final = 'C:/Users/Frank/Desktop/New_DHT11_Data_Collection.xlsx'
with pd.ExcelWriter(output_file_path_final) as writer:
    sensors.to_excel(writer, sheet_name='Sensors', index=False)  # Save Sensors
table
    measurements.to_excel(writer, sheet_name='Measurements', index=False)  # Save
Measurements table
    grouped.to_excel(writer, sheet_name='Statistics', index=False)  # Save
Statistics table

print(f"Data has been saved to {output_file_path_final}")  # Output the success
message
print(f"Data has been saved to {output_file_path_final}") # Print the path of the
saved file
```

First, we previously created a single table, but now we need more than one table. To establish relationships between them, we will create two tables. One table, called the ‘Sensors’ table, is used to store all our sensors, as we will be using DHT11, DHT22, and MAX30102. Therefore, the sensors need to have a separate table.

The other table is the ‘Measurements’ table. In the last lesson, we mainly modified the ‘Measurements’ table. We will then need to establish the relationship between the two tables, linking different sensors to their corresponding measurements.

```
# Create the `sensors` table
# Select the 'Sensor ID' and 'Location' columns from the DataFrame, remove duplicate rows, and reset the index.
sensors = df[['Sensor ID', 'Location']].drop_duplicates().reset_index(drop=True)

sensors['Sensor ID'] = sensors['Sensor ID'].astype(str)  # Convert the data type of the 'Sensor ID' column to string.
sensors['Location'] = sensors['Location'].astype(str)  # Convert the data type of the 'Location' column to string.
sensors['Sensor Key'] = sensors.index + 1  # Add a new column 'Sensor Key' to the `sensors` table, with values being the row index plus 1.
```

df[['Sensor ID', 'Location']]: Create a new DataFrame by selecting the columns 'Sensor ID' and 'Location' from the original DataFrame

.drop_duplicates(): Remove duplicate rows. For example, if there are two rows with the same 'Sensor ID' and 'Location', one row will be kept and the other will be deleted

.reset_index(drop=True): After removing duplicate rows, reset the index so that it starts at 0 consecutively and the old index column is not retained.

.astype(str): This method is used to convert the data type of a column. Here, it is used to ensure that the data type of the Sensor ID and Location columns is a string (str).

sensors.index: Get the index of the current sensors DataFrame. By adding 1 to this index, we create a new column Sensor Key starting at 1 to assign a unique serial number to each sensor.

Output

	A	B	C
1	**Sensor ID**	**Location**	**Sensor Key**
2	DHT11	bedroom	1
3	DHT22	balcony	2
4	DHT11	First Floor	3
5	DHT22	bathroom	4
6	DHT11	backyard	5

```
# Create the `measurements` table
measurements = df.drop(columns=['Location']).copy()  # Copy the DataFrame and
remove the 'Location' column.
measurements = measurements.merge(sensors, on='Sensor ID', how='left')  # Merge
the `sensors` table into the `measurements` table based on 'Sensor ID'.
measurements = measurements.drop(columns=['Sensor ID'])  # Remove the 'Sensor ID'
column.
measurements = measurements.rename(columns={'Sensor Key': 'Sensor ID'})  # Rename
the 'Sensor Key' column to 'Sensor ID'.
```

Merge the above sensors table to add Sensor Key and Location by removing the redundant Location column. Then remove the original Sensor ID column and rename the Sensor Key column to Sensor ID, finally creating a clean measurements table with clear unique identifiers.

Create a new column and divide the comfort index level into three parts: high, medium and low. The goal is to divide it into three different levels to express different comfort environments. When it is low, it corresponds to the lowest 33% of the comfort level, medium is 33%-66%, and high is the rest. So cut it into three parts and then label it low, medium, and high accordingly.

pd.cut is a function in the Pandas library that is used to segment data. It is often used to split continuous data into discrete bins and assign labels to these bins.

- **measurements['Comfort Index'] The data column to be segmented, i.e. the Comfort Index column.**
- **The bins parameter defines the boundaries of the bins.**
- **The labels parameter assigns a label to each bin.**